Northern University, Nowshera

Association, Aggregation & composition

Week # 12 - Lecture 23- 24

Northern

Spring 2024

Learning Objectives:

1. **Assignment Solution (Week 11)**
2. **Association**
3. **Aggregation**
4. **Composition**
5. **UML representation of Association, Composition and Aggregation**
6. **Association vs Composition vs Aggregation with example**
7. **Key Points - Conclusion**
8. **Week 12 Assignment**

Northern

1. Assignment Solution

Q: Write code for the following classes.

Person Class: Animal class has attributes: String name, address and int age. Write setperson() function to set values and getPerson() to Print attributes. Also write appropriate constructors.

Employee Class: Write another class Employee having attributes department and salary of type string and double. Write methods setEmployee(), getEmployee() and appropriate constructors for Employee class.

Student Class: Write a class Student having attributes registration number and GPA of type string and float. Also write setStudent(), getStudent() methods and required constructors.

Use the concept of inheritance to achieve the above functionality. Write a main() function to display the information of employee and student.

Note: Call the constructors/methods of parent class in child class where required.

Solution:

```
class Person
{
    protected String name, address;
    protected int age;

    public Person(){}
    public Person(String name, String address, int age)
    {
        this.name=name;
        this.address=address;
        this.age=age;
    }
    void setPerson()
    {
        Scanner input=new Scanner(System.in);
        System.out.println("Enter Your Name :");
        this.name=input.nextLine();
        System.out.println("Enter Your Address :");
        this.address=input.nextLine();
        System.out.println("Enter Your Age :");
        this.age=input.nextInt();
```

```
    }
    void getPerson()
    {
        System.out.print(this.name+"\t"+this.address+"\t"+this.age+"\t");

    }
}
class Students extends Person
{
    private String regNo;
    private float CGPA;
    public Students()
    {
        super();
    }
    public Students(String name, String address, int age, String regNo, float CGPA)
    {
        super(name,address,age);
        this.regNo=regNo;
        this.CGPA=CGPA;
    }
    void setStudent()
    {
        super.setPerson();
        Scanner input=new Scanner(System.in);
        System.out.println("Enter Your Registration Number :");
        this.regNo=input.nextLine();
        System.out.println("Enter Your CGPA :");
        this.CGPA=input.nextFloat();
    }
    void getStudent()
    {
        super.getPerson();
        System.out.println(this.regNo+"\t"+this.CGPA);

    }
}
class Employee extends Person
{
    private String department;
    private double salary;
    public Employee()
    {
        super();
    }
    public Employee(String name, String address, int age, String department, float
salary)
    {
        super(name,address,age);
        this.department=department;
        this.salary=salary;
    }
```

```
    void setEmployee()
    {
        super.setPerson();
        Scanner input=new Scanner(System.in);
        System.out.println("Enter Your Departmant :");
        this.department=input.nextLine();
        System.out.println("Enter Your Salary :");
        this.salary=input.nextDouble();
    }
    void getEmployee()
    {
        super.getPerson();
        System.out.println(this.department+"\t"+this.salary);

    }
}
class MainClass
{
    public static void main(String [] args)
    {
        Students s=new Students();
        System.out.println("Enter details of student: ");
        s.setStudent();
        s.getStudent();

        Students s2=new Students("Adeel", "Rawalpindi", 21, "2019-arid-1234",2.5f);
        s2.getStudent();

        Employee e=new Employee();
        System.out.println("\n\nEnter details of employee: ");
        e.setEmployee();
        e.getEmployee();
    }

}
```

Output

```
Enter details of student:

Enter Your Name :
Ali Ahmed
Enter Your Address :
Rawalpindi
Enter Your Age :
23
Enter Your Registration Number :
18-arid-1234
Enter Your CGPA :
2.6
```

```
Ali Ahmed        Rawalpindi       23       18-arid-1234     2.6
Adeel    Rawalpindi       21       2019-arid-1234   2.5


Enter details of employee:
Enter Your Name :
Sohail ahemed
Enter Your Address :
Islamabad
Enter Your Age :
32
Enter Your Departmant :
Computer Science
Enter Your Salary :
150000
Soahail ahemed   Islamabad        32       Computer Science 150000.0
```

Northern

2. Association

Association is relation between two separate classes which establishes through their Objects. In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. This relationship between two objects is known as the *association*. Association is a relationship where all objects have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently. **Composition** and **Aggregation** are the two forms of association.

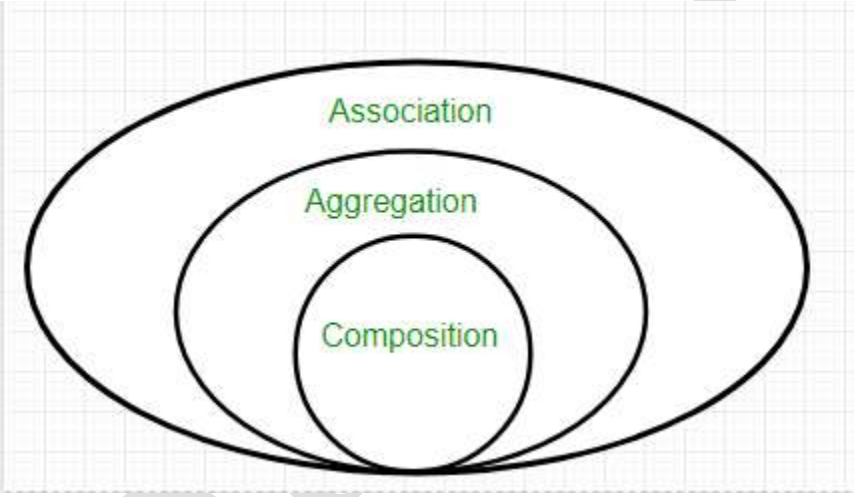


Figure 1: Association, Aggregation and Composition

Example 1: Bank and employee class → Association

```
// Java program to illustrate the
// concept of Association

class Bank {
    private String name;

    // bank name
    Bank(String name) {
        this.name = name;
    }

    public String getBankName() {
        return this.name;
    }
```

```
}

// employee class
class Employee {
    private String name;

    // employee name
    Employee(String name) {
        this.name = name;
    }

    public String getEmployeeName() {
        return this.name;
    }
}

// Association between both the classes in main method
class Association {
    public static void main(String[] args) {
        Bank bank = new Bank("BOP");
        Employee emp = new Employee("Sadia");

        System.out.println(emp.getEmployeeName() +
                " is employee of " + bank.getBankName());
    }
}
```

OUTPUT

```
Sadia is employee of BOP
```

In above example two separate classes Bank and Employee are associated through their Objects. Note that there is no is-a relationship between bank and employee so inheritance is not suitable here. Bank can have many employees (has-a relationship), so it is a one-to-many relationship as shown in figure 2.

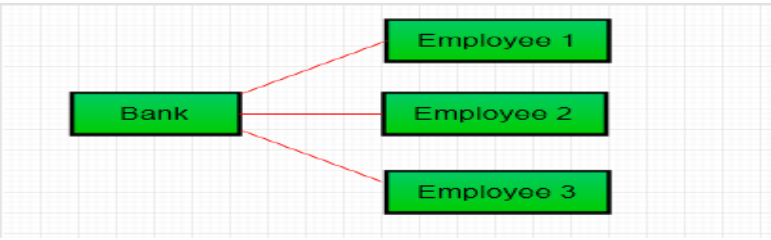


Figure 2: Bank has employees

3. Aggregation

Aggregation is a specialized form of Association where all objects have their own lifecycle but there is ownership, like a Player which is part of a Team, can exist without a team and can become part of other teams as well. It is a special form of Association where:

- It represents **Has-A** relationship.
- It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, **both the entries can survive individually** which means ending one entity will not affect the other entity

Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, teacher object will not be destroyed. We can think about it as a "has-a" relationship.
Another example of **Aggregation** is Student in School class, when School closed, Student still exist and then can join another School or so. In UML notation, aggregation is denoted by an **empty diamond**.

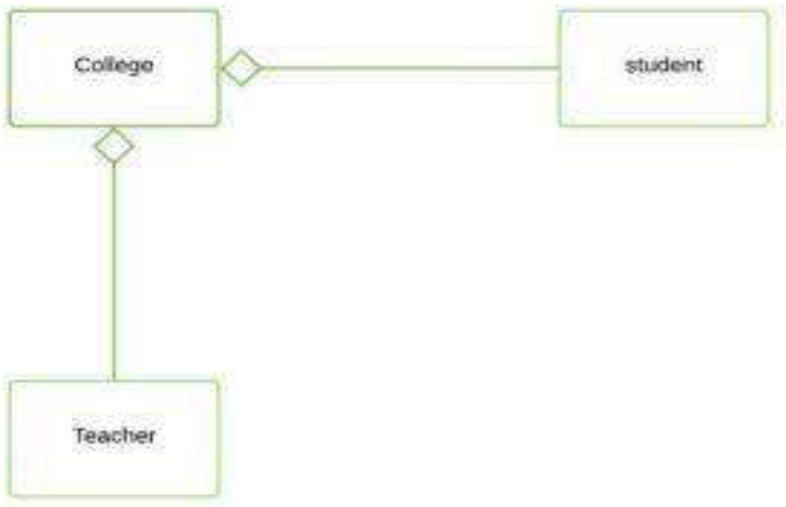
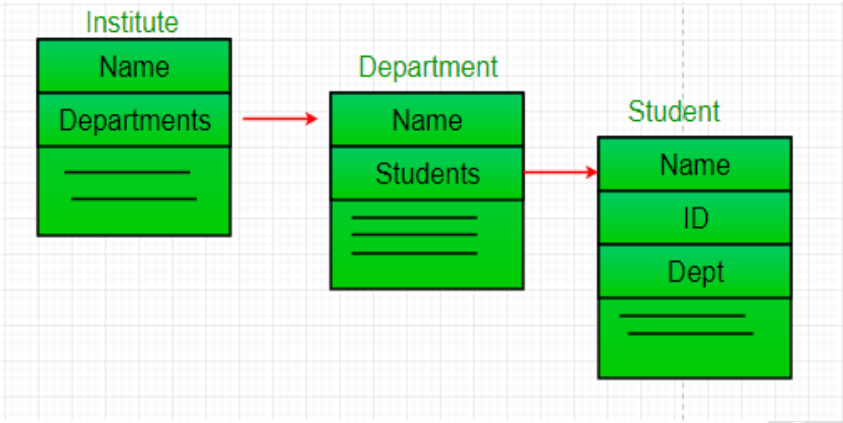


Figure 3: Aggregation

In this example, there is an Institute which has no. of departments like CS, IT. Every department has number of students. So, we will make an Institute class which has a reference to Object or number of Objects of the Department class. That means Institute class is associated with

Department class through its Objects, and Department class has also a reference to Object or Objects of Student class, so it is associated with Student class through its Object(s).



Example 2: Institute, department & student class → Aggregation

```
// Java program to illustrate
//the concept of Aggregation.
// student class
class Student {
    String name;
    int id;
    String dept;

    Student(String name, int id, String dept) {

        this.name = name;
        this.id = id;
        this.dept = dept;

    }
    static void printStudents(Student students[])
    {
        for(Student s : students)
        {
            System.out.println(s.name+"\t"+s.id+"\t"+s.dept);
        }
    }
}

/* Department class contains list of student
Objects. It is associated with student
class through its Object(s). */
class Department {

    String name;
```

```
    private Student stds[];

    Department(String name, Student stds[]) {

        this.name = name;
        this.stds = stds;
    }

    public Student[] getStudents() {
        return stds;
    }
}

/* Institute class contains list of Department
Objects. It is asoociated with Department
class through its Object(s).*/
class Institute {

    String instituteName;
    private Department[] departments;

    Institute(String instituteName, Department departments[]) {
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // count total students of all departments
    // in a given institute
    public int getTotalStudentsInInstitute() {
        int noOfStudents = 0;
        Student[] students=new Student[4];
        for (Department dept : departments) {
            students = dept.getStudents();
            for (Student s : students) {
                noOfStudents++;
            }
        }
        return noOfStudents;
    }

}

// main method
class MainClass {
    public static void main(String[] args) {
        Student s1 = new Student("Alia ", 1, "CS");
        Student s2 = new Student("Asad ", 2, "CS");
        Student s3 = new Student("Sadia", 1, "IT");
        Student s4 = new Student("Sadaf", 2, "IT");

        // making an array of CS Students.
        Student CS_students[]=new Student[2];
```

```
        CS_students[0]=s1;
        CS_students[1]=s2;

        // making an array of IT Students
        Student IT_students[]=new Student[2];
        IT_students[0]=s3;
        IT_students[1]=s4;

        Department CS_dept = new Department("CS", CS_students);
        Department IT_dept = new Department("IT", IT_students);


        Department departments[]=new Department[2];
        departments[0]=CS_dept;
        departments[1]=IT_dept;

        // creating an instance of Institute.
        Institute institute = new Institute("BIIT", departments);

        System.out.print("Total students in institute: ");
        System.out.print(institute.getTotalStudentsInInstitute());

        Student.printStudents(CS_students);
        Student.printStudents(IT_students);

    }
}
```

OUTPUT

```
Total students in institute: 4
Alia   1       CS
Asad   2       CS
Sadia  1       IT
Sadaf  2       IT
```

4. Composition

Composition is again specialized form of Aggregation and we can call this as a “death” relationship. Child object does not have its lifecycle and if parent object is deleted all child objects will also be deleted. Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

- It represents **part-of** relationship.
- In composition, both the entities are dependent on each other.
- When there is a composition between two entities, the composed object **cannot exist** without the other entity.

Let's take an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belong to two different houses. If we delete the house - room will automatically be deleted. In UML notation, a composition is denoted by a **filled diamond**. Let's take example of **Library**.

Example 3: Books & Library class → Composition

```
// Java program to illustrate
// the concept of Composition

// class book
class Book {

    public String title;
    public String author;

    Book(String title, String author) {

        this.title = title;
        this.author = author;
    }
}

// Libary class contains
// list of books.
class Library {
private String libName;
    // reference to refer to list of books.
    private final Book book;

    Library(String libName, String title, String Author) {
        book=new Book(title,Author);
        this.libName = libName;
    }

    public Book getLibraryBook() {

        return book;
    }

}
```

```
// main method
class MainCl {
    public static void main(String[] args) {

     Library library = new Library("City Library","EffectiveJ Java", "Joshua Bloch");

     Book bk = library.getLibraryBook();
     System.out.println("Book Title " + bk.title + " and "
                    + "Author is " + bk.author);
     }

    }
```

OUTPUT

```
Book Title EffectiveJ Java and Author is Joshua Bloch
```

In above example a library can have no. of **books** on same or different subjects. So, If Library gets destroyed then All books within that particular library will be destroyed. i.e. book cannot exist without library. That's why it is composition.

Conclusion:

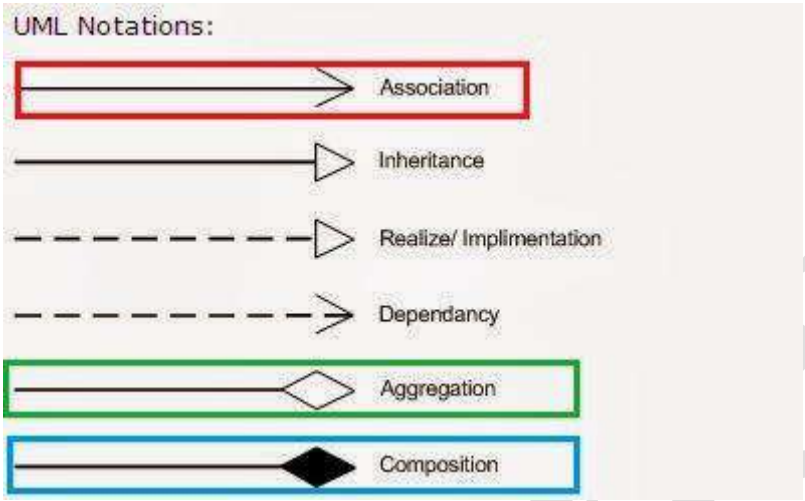
The composition is stronger than Aggregation. In Short, a relationship between two objects is referred as an association, and an association is known as composition when one object *owns* other while an association is known as aggregation when one object uses another object.

5. UML representation of Association, Composition and Aggregation

UML has different notations to denote aggregation, composition and association.

- Association is denoted by the simple arrow
- while aggregation is denoted by empty diamond-head arrow and
- composition is denoted by filled diamond-head arrow.

When you draw UML diagram for two related class A and B, where A is associated with B then its denoted by A -> B. Similar way is used to show aggregation and composition between two classes. Here are UML notations for different kind of dependency between two classes.



As I said all three denotes relationship between object and only differ in their strength, you can also view them as below, where composition represents strongest form of relationship and association being the most general form.

6. Difference with example

Example 4: Difference

```
// Java program to illustrate the difference between Aggregation
// Composition.

// Engine class which will be used by car.
//so 'Car'class will have a field of Engine type.

class Engine
{
    // starting an engine.
    public void work()
    {

        System.out.println("Engine of car has been started ");
    }
```

```
}

class Car
{

    // For a car to move,
    // it need to have a engine.
    private final Engine engine; // Composition
    //private Engine engine;    // Aggregation

    Car(Engine engine)
    {
        this.engine = engine;
    }

    // car start moving by starting engine
    public void move()
    {

        //if(engine != null)
        {
            engine.work();
            System.out.println("Car is moving ");
        }
    }
}

class MainClass
{
    public static void main (String[] args)
    {

        // making an engine by creating an instance of Engine class.
        Engine engine = new Engine();

        Car car = new Car(engine);
        car.move();

    }
}
```

OUTPUT

```
Engine of car has been started
Car is moving
```

7. Key Points

Here is the list of differences between Composition and Aggregation in point format, for quick review. As I said the key difference between them comes from the point that in the case of Composition, One object is OWNER of another object, while in the case of aggregation, one object is just a USER or another object.

- If A and B two classes are related to each other such that, B ceased to exist, when A is destroyed, then the association between two objects is known as Composition. An example is Car and Engine. While if A and B are associated with each other, such that B can exist without being associated with A, then this association in known as **Aggregation**.

- In the case of **Composition** A owns B e.g. Person is the owner of his Hand, Mind and Heart, while in the case of Aggregation, A uses B e.g. Organization uses People as an employee.

- In UML diagram Association is denoted by a normal arrow head, while Composition is represented by filled diamond arrow head, and Aggregation is represented by an empty diamond arrow head, As shown in below and attached diagram in the third paragraph.

  Association A---->B
  Composition A-----<filled>B
  Aggregation A-----<>B

- Aggregation is a lighter form of Composition, where a sub-part object can meaningfully exist without main objects.

- In Java, you can use final keyword to represent Composition. Since in Composition, Owner object expects a part object to be available and functions, by making it final, your

Northern

provide guarantee that, when Owner will be created, this part object will exist. This is actually a *Java idiom to represent a strong form of association* i.e. composition between two objects.

Northern

Assignment

Q: Write code for the following classes.

You are required to implement a system where information of authors with their books has been stored using the concept discussed in this lesson (Association, aggregation and composition).

Author class contains author name (String), email (String), total number of written books with following book detail.

Book class has book ID (String), Book Title (String), price (float) and publisher (String) has attributes.

Associate these two classes with each other (according to your understanding) in a way that user can get the information of author with detail of his total written books, like book title, publisher, id and price. Demonstrate your program in main() function by creating an object of class author with at least 3 books.

Note: Write setter and getter functions for both classes and also write appropriate constructors.

Northern